Byte code Interpreter for 8051 Microcontroller

N. Jeenjun S. Khuntaweetep and S. Somkuarnpanit

***Abstract*— This paper proposes a design of byte code interpreter for 8051 microcontroller. We developed a program based on Java language *"write once, run everywhere"*. It can be employed to operate with another system since the program interfaces directly to the interpreter, instead of the real system. There are two instruction sets provided in the program. The first is condition commands such as IF and ELSE. The second is the commands for peripheral interfacing such as I^2C, RS232 interface, for parallel port, timer and LCD interface. Interpreter is developed with C language, due to its simple structure. It is easy to develop and to modify the code. From the results, condition commands and no condition commands can be execute correctly.**

***Index Terms*— Byte code, Interpreter, Byte code.**

I. Introduction

To use microcontroller, user has to understand its architecture and language. Microcontroller of each family has its own architecture and instruction sets. For example PIC family is RISC and 8051 family is CISC. When user wants to change microcontroller to the other, user has to develop the new program to support new microcontroller. It is waste of time because user has to study architecture and instruction sets of new microcontroller as well.

Interpreter for microcontroller may be the alternative way to solve the problem. When user wants to change the new microcontroller, user only installs interpreter to new microcontroller but do not write the new program.

Example for byte code is Java language.

The Java as introduced by Sun Microsystems [1] in 1994 has spread throughout the computer industry and has reached all domains. As good as Java is for providing *"write once, run everywhere"* software. In figure 1, programmer writes Java program. Java compiler will compile Java program to Java byte code. To use Java program, user has to install Java virtual Machine [2] to operating system target. In case user wants to change to new operating system, users just installs Java virtual Machine and Java program can be ran as normal. For this case Java virtual Machine is interpreter of Java system.

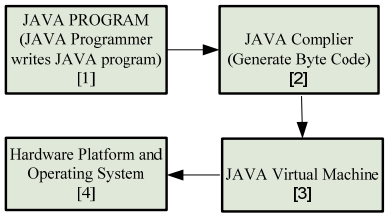

Figure 1. Java operation

The favorite toy in the world, Lego mind storm [3]. It has H8 microcontroller with Lego interpreter inside. User interface is based on graphic block. It is easy for beginner even children can develop their own applications.

For the other examples of interpreter are MetaCricket [4] developed by MIT, GOGO BOARD [5], BasicStamp by Parallax Inc. [6], picoJava-I [7] picoJava-II [8] by Sun Microsystems, An Embedded Java Virtual Machine [9][10], The Byte code Firmware Design for Microcontroller Device [11].

II. System Comparison

Standard Microcontroller (MCU) language design is depicted in figure 2, user writes assembly program to target MCU and compiles to machine code of each system. In this case, user has to develop 2 programs for both 8051 and PIC.

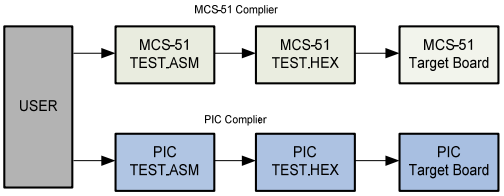

Figure 2. Standard MCU

In figure 3, user installs byte code interpreter into both 8051 and PIC microcontroller then writes the program with byte code instruction sets. In this case user writes the program only 1 time.

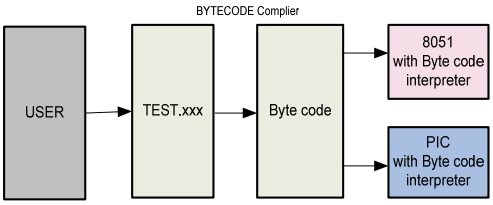

Figure 3. MCU with byte code interpreter

Manuscript received December 27, 2009.

N. Jeenjun is a graduate student in Master Degree in the Faculty of Engineering at King Mongkut's Institute of Technology, Ladkrabang, Bangkok Thailand.

S. Khuntaweetep is now with the Department of Electronics, Faculty of Engineering at King Mongkut's Institute of Technology, Ladkrabang, Bangkok Thailand (e-mail: kksuchar@kmitl.ac.th).

S. Somkuarnpanit is with the Department of Electronics, Faculty of Engineering at King Mongkut's Institute of Technology, Ladkrabang, Bangkok Thailand (e-mail: kssuripo@kmitl.ac.th).

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

III. Interpreter Diagram

The interpreter for MCU has 3 parts, they are depicted in figure 4.

- VM Controller is the core of interpreter, it controls fetch/execute and condition commands, IF, ELSE, REPEAT and GOTO.
- VM EXECUTE is byte code decoder.
- Peripheral is hardware modules. It consists of, Parallel Port, Serial Port, I^2C Port, Timer, LCD Port and Arithmetic functions.

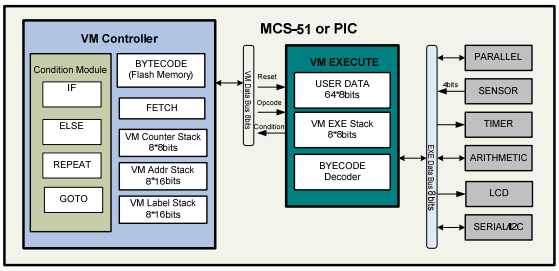

Figure 4. Interpreter diagram

The system flow chart is depicted in figure 5. After MCU is reset, interpreter fetches byte code from built in Flash memory. Condition1 will be checked, whatever it is normal task or condition task. If it is normal task, next byte code will be decoded as normal. If not, interpreter will check condition before go to next command. After the operation is done, program counter (PC) will be increased to next line address.

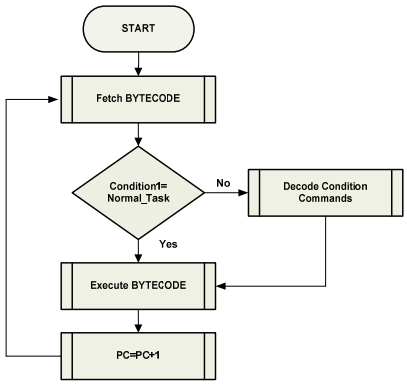

Figure 5. Interpreter flow chart

A. Data Stack

Interpreter is designed to have 16 levels of data stack. The advantage of data stack is, store the result of condition command IF/ELSE and WAITUNITL and the result from arithmetic commands.

Data stack example is depicted in figure 6.

00,02,09,01,00,06,00,01,40,01,35,1C,0A

- 00,02 : Line address of program
- 09 : SUM
- 01 : Condition1 (IF statement)
- 00 : No Condition2
- 06 : Number slave of IF statement
- 00 : This is master line
- 01,40 : Push data 40H to current stack
- 01,35 : Push data 35 to next stack
- 1C : Equal comparison byte code
- 0A : Comparison data in stack and stack+1, the result will be in stack

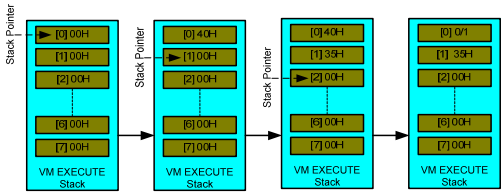

Figure 6. Data stack

B. The byte code format

The byte code format consists of 7 parts, it is depicted in figure 7

- AddrH and AddrL are 16 bits line address. (Maximum is 65,535 lines)
- SUM is used byte in each line (AddrL, AddrH and SUM are not included)
- Condition1 is the primary condition.
- Condition2 is the secondary condition.
- CondListH, in case if Condition1 is not 00H
  - Equal 00H : It is slave line address.
  - Not equal 00H : It is master line address.
- CondListL, in case if Condition1 is not 00H
  - Equal 00H : It is master line address.
  - Not equal 00H : It is slave line address.
- Byte code is 8 bits command.

Figure 7. Byte code format

C. Byte code Instruction sets

The instruction sets are categorized to 2 sections:

No condition commands

For no condition commands, Condition1 will be 00H, Condition2 CondListH and CondListL will not be appeared in the line. Format is depicted in figure 8. Example for this commands are TX232, RX232, I^2CSTART, I^2CWRITE, LCD, WAITTIME, OUTPORT and INPORT.

Figure 8. No condition command format

Condition commands

For condition commands consist of, IF, ELSE, REPEAT and GOTO. The format is depicted in figure 9. Condition1 will not be 00H and Condition2 is 00H reserved for further use.

Figure 9. Condition command format

Interpreter is designed to have 8 levels of IF and ELSE commands. In figure 10 shows 3 levels of IF command and 2 levels of ELSE command.

For IF command, byte code 01H-08H are reserved and 81H-88H are reserved for end of IF line. ELSE command, byte code 09H-0FH are reserved and 89H-8FH are reserved for end of ELSE line.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
IF ()                           ;Condition1 = 01H (IF Level 1)
{
        IF()                    ;Condition1 = 02H (IF Level 2)
        {
                IF()            ;Condition1 = 03H (IF Level 3)
                {
                }               ;Condition1 = 83H (Close IF Level 3)
                ELSE()          ;Condition1 = 0BH (ELSE Level 3)
                {
                }               ;Condition1 = 8BH (Close ELSE Level 3)
        }                       ;Condition1 = 82H (Close IF Level 2)
        ELSE()                  ;Condition1 = 0AH
        {
        }                       ;Condition1 = 8AH (Close ELSE Level 2)
}                               ;Condition1 = 81H (Close IF Level 1)
ELSE ()                         ;Condition1 = 09H (ELSE Level 1)
{
}                               ;Condition1 = 89H (Close ELSE Level 1)
```

Figure 10. Example IF/ELSE commands

For REPEAT command, interpreter is designed to have 8 levels. Number of repeating can be 0-255. Byte code 21H-28H is reserved for REPEAT command.

GOTO command, it must have label to go to. Label or address can be 0000H-FFFFH. Byte code 80H is reserved for GOTO command.

All of commands will be explained in testing result section.

D. Memory allocation

Memory allocation is depicted in figure 11, the first 16KB is reserved for interpreter installation and the second is user code area is 16KB Flash or 32KB EEPROM as configuration setting. In this paper, user code area is in 16KB Flash memory.

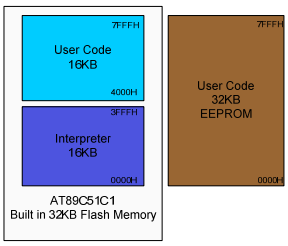

Figure 11. Memory allocation

E. Hardware

Hardware for testing the system is ETT-JR51USB [12] below is specification of hardware.

- AT89C5131 MCU with 6 clocks running
- 32KByte Flash and 1Kbyte Sram
- USB cable for downloading program
- 1xUART with maximum 11520 bps
- 1x I^2C port
- 5VDC power supply on board

IV. Testing Results

We chose some of byte code examples for testing the interpreter. It consists of I^2C, RS232, LCD and Parallel Port with delay time.

The first example code is depicted in figure 12, the program will produce I^2C start signal, write data to I^2C with 5AH then produce I^2C stop signal. The testing result is depicted in figure 13.

```
User Code                       BYTECODE

I2CSTART                ;       00,00,02,00,4E,
I2CWRITE(0x5A)          ;       00,01,04,00,01,5A,4A,
I2CSTOP                 ;       00,02,02,00,4F,
```

Figure 12. Example program#1

The first line 00,00,02,00,4E

- 00,00 : Line address
- 02 : SUM
- 00 : Condition1 (No condition command)
- 4E : Write start I^2C signal

The second line 00,01,04,00,01,5A,4A

- 00,01 : Line address
- 04 : SUM
- 00 : Condition1 (No condition command)
- 01,5A : Push data 5AH to data stack
- 4A : Send data in data stack to I^2C

The third line 00,02,02,00,4F

- 00,02 : Line address
- 02 : SUM
- 00 : Condition1 (No condition command)
- 4F : Write stop I^2C signal

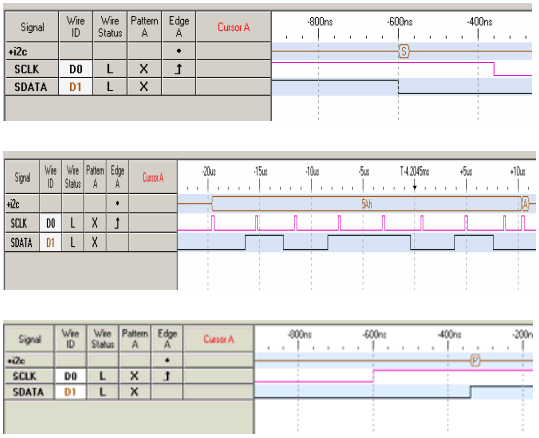

Figure 13. I^2C Start, Write data and Stop signals

The second example is depicted in figure 14. The program receives data from RS232 channel and sends the same data back to host (Personal computer). Figure 15 shows RS232 signals.

The first line 00,00,06,64,00,00,05,00,01

- 00,00 : Line address
- 06 : SUM
- 64 : Condition1 (Label command)
- 00 : No Condition2
- 00,05 : Label stack location
- 00,01 : Line address for go to command

The second line 00,01,02,00,14

- 00,01 : Line address
- 02 : SUM
- 00 : Condition1 (No condition command)
- 14 : Read data from RS232

The third line 00,02,02,00,04,13

- 00,02 : Line address
- 02 : SUM
- 00 : Condition1 (No condition command)
- 04 : Push RS232 data to data stack
- 13 : Write data from data stack to RS232

The fourth line 00,03,05,65,00,00,05,00

- 00,03 : Line address
- 05 : SUM
- 65 : Condition1 (Go to command)
- 00 : No Condition2
- 00,05 : Reload Label at stack location 05
- 00 : Reserved for further used

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
User Code                          BYTECODE

LABEL:                             ;00,00,06,64,00,00,05,00,01,
RX232                              ;00,01,02,00,14,
TX232(USERTEMP)                    ;00,02,03,00,04,13,
GOTO LABEL                         ;00,03,05,65,00,00,05,00,
```

Figure 14. Example program#2

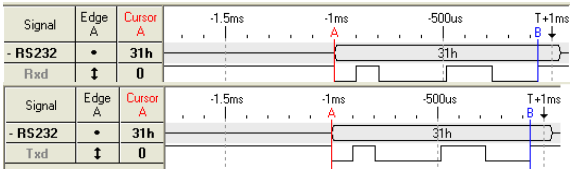

Figure 15. TX232 and RX232

The third example is depicted in figure 16, the program will send data to parallel port with data 1, 2, 4, 8, 10H, 20H, 40H and 80H with 400mS delay time. Testing result is depicted in figure 17.

The first line 00,00,06,64,00,00,05,00,01

- 00,00 : Line address
- 06 : SUM
- 64 : Condition1 (Label command)
- 00 : No Condition2
- 00,05 : Label stack location
- 00,01 : Line address for go to command

The second line 00,01,04,00,01,01,4C

- 00,01 : Line address
- 04 : SUM
- 00 : Condition1 (No condition command)
- 01,01 : Push data 01 to data stack
- 4C : Write data in data stack to parallel port

The third line 00,02,04,00,01,04,10

- 00,02 : Line address
- 04 : SUM
- 00 : Condition1 (No condition command)
- 01,04 : Push data 01 to data stack
- 10 : Delay time with data in data stack multiply by 100mS

The fourth line 00,03,04,00,01,02,4C

- 00,03 : Line address
- 04 : SUM
- 00 : Condition1 (No condition command)
- 01,02 : Push data 02 to data stack
- 4C : Write data in data stack to parallel port

The rest program will be repeated the same as line1 and line2 but only change data out.

```
User Code                          Byte Code

LABEL:                             ;00,00,06,64,00,00,05,00,01,
OUTPORT(0x01)                      ;00,01,04,00,01,01,4C,
WAITTIME(4)                        ;00,02,04,00,01,04,10,
OUTPORT(0x02)                      ;00,03,04,00,01,02,4C,
WAITTIME(4)                        ;00,04,04,00,01,04,10,
OUTPORT(0x04)                      ;00,05,04,00,01,04,4C,
WAITTIME(4)                        ;00,06,04,00,01,04,10,
OUTPORT(0x08)                      ;00,07,04,00,01,08,4C,
WAITTIME(4)                        ;00,08,04,00,01,04,10,
OUTPORT(0x10)                      ;00,09,04,00,01,10,4C,
WAITTIME(4)                        ;00,0A,04,00,01,04,10,
OUTPORT(0x20)                      ;00,0B,04,00,01,20,4C,
WAITTIME(4)                        ;00,0C,04,00,01,04,10,
OUTPORT(0x40)                      ;00,0D,04,00,01,40,4C,
WAITTIME(4)                        ;00,0E,04,00,01,04,10,
OUTPORT(0x80)                      ;00,0F,04,00,01,80,4C,
WAITTIME(4)                        ;00,10,04,00,01,04,10,
GOTO LABEL                         ;00,11,05,65,00,00,05,00,
```

Figure 16. Example program#3

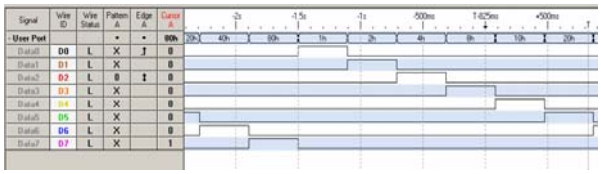

Figure 17. Parallel Port

The last example is LCD command, figure 18 is program and figure 19 is testing result.

The first line 00,00,02,00,A2

- 00,00 : Line address
- 02 : SUM
- 00 : Condition1 (No condition command)
- A2 : Initial LCD byte code command

The second line

00,01,19,00,01,31,A0,01,32,A0,01,33,A0,01,34,A0,01,35,A0,01,36,A0,01,37,A0,01,38,A0,

- 00,01 : Line address
- 19 : SUM
- 00 : Condition1 (No condition command)
- 01,31 : Push data 31(character “1”) to data stack
- A0 : Write data in data stack to LCD
- 01,32 : Push data 32(character “2”) to data stack
- A0 : Write data in data stack to LCD
- 01,33 : Push data 33(character “3”) to data stack
- A0 : Write data in data stack to LCD
- 01,34 : Push data 34(character “4”) to data stack
- A0 : Write data in data stack to LCD

The rest program will write data “5678abcdefgh” to LCD.

```
[0]   lcdinit
[1-2] lcd('12345678abcdefgh')
[3]   loop:
[4]   goto loop;

[0] 00,00,02,00,A2,
[1] 00,01,19,00,01,31,A0,01,32,A0,01,33,A0,01,34,A0,01,35,A0,01,36,A0,01,37,A0,01,38,A0,
[2] 00,01,1A,00,A5,01,41,A0,01,62,A0,01,63,A0,01,64,A0,01,65,A0,01,66,A0,01,67,A0,01,68,A0,
[3] 00,03,06,64,00,00,05,00,01,
[4] 00,04,05,65,00,00,05,00
```

Figure 18. LCD program

Figure 19. LCD output

A. Result analysis: Memory usage

This paper we have installed byte code interpreter into 8051(CISC) microcontroller. The interpreter is written by C language with KEIL compiler. Table4.1 is comparison for memory usage.

TABLE I. TABLE4.1 MEMORY USAGE

MCU	IDE	Code	Data
AT89C5131	KEIL	10.97KB	434B
PIC18F4620	MPLAB	14.96KB	450B

B. Result analysis: Speed improvement

From the previous paper, the byte code firmware for Microcontroller Device [11], user program was stored in

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

external memory EEPROM with I^2C protocol. It is the bottle neck of the system because it took 10 clocks to fetch data from EEPROM 1 Byte. In this paper we have improved the problem by use built in Flash memory of the microcontroller, it took only 2 clocks to fetch data 1 Byte.

V. Conclusions

From testing result, even we changed system from PIC to 8051 microcontroller: the interpreter still can execute byte code commands as correctly. It is proven the independent hardware conception. The execution byte code command is improved by changing external memory to internal Flash memory. In the further: to reduce interpreter code size and speed improvement, we can use Assembly language instead of C language. Finally: to check byte code syntax, the complier should be developed as well.

References

[1] S. Wilson and J. Kesselman, *"Java platform perform ance strategies and tactics"*, Addison-Wesley, Boston, 2000.

[2] T. Lindholm and F. Yellin, *"The Java virtual machine specification"*, Addison-Wesley, Reading, Mass., 1997.

[3] Jin Sato, *"LEGO MINDSTORMS: The Master's Tech nique"*, O'Reilly Media, Inc.,2008.

[4] F.Matin, B.Mikhak, B.Silverman, "*MetaCricket: A de signner's kit for making computational devices*", IBM Systems Journal, Vol. 39, NOS 3&4, 2000.

[5] Arnan (Roger) Sipitakiat, "*GOGO BOARD*", Available from: http://www.gogoboard.org

[6] Al Williams, "*Microcontroller Projects Using the Basic Stamp*", CMP Books, 2002.

[7] Sun Microsystems, *"picoJava-I: picoJava-I Core Micro processorArchitecture"*, Sun Microsystems white paper, October 1996.

[8] Sun Microsystems, *"picoJava-II: Java Processor Core"*, SunMicrosystems data sheet, April 1998.

[9] Graham Mathias, Kenneth B. Kent "*An Embedded Java Virtual Machine Using Network-on-Chip Design*", IEEE Int. Workshop on Rapid System Prototyping, 2006.

[10] Kenneth B. Kent, Micaela Serra, "*Hardware/Software Co-Design of a Java Virtual Machine*", IEEE, 2000.

[11] Narakorn Jeenjun, "*The Byte code Firmware Design for Microcontroller Device*", KKU Journal, Vol 34, Page 535-546, 2007.

[12] ETT-JR51USB Board Available from: www.ett.co.th

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)